

Trabalho prático 2

Redes de Computadores

Rede de Computadores

Trabalho realizado por:

Nuno Oliveira [up201806525@fe.up.pt](mailto:up201806525@fe.up.pt)

Luís Pinto [up201806206@fe.up.pt](mailto:up201806206@fe.up.pt)

**Sumário**

Este relatório descreve a arquitetura da aplicação de download que implementa um cliente simples do protocolo FTP como descrito no RFC959 por nós desenvolvida bem como os resultados da mesma. Para além disso, fizemos também uma análise sobre a série de experiências relativas à segunda parte deste segundo trabalho prático.

**Introdução**

Para o segundo trabalho prático de Redes de Computadores foi-nos proposto o desenvolvimento de uma aplicação que faz download de um único ficheiro implementado um cliente FTP bem como a configuração de uma rede e análise da mesma. Os objetivos deste trabalho são entender o comportamento de FTP, usar sockets e TCP na linguagem C, perceber DNS do serviço e usá-lo com o programa cliente e, por fim, entender a arquitetura de uma rede de computadores.

O relatório visa então explicar tudo o que foi mencionado sendo composto pelas seguintes secções:

* Aplicação de download
  + Arquitetura
  + Resultados de dum download com sucesso
* Configuração e análise da rede
  + Experiência 1 – configurar uma rede IP
  + Experiência 2 – implementar duas virtual LANs num switch
  + Experiência 3 – configurar um router em Linux
  + Experiência 4 – configurar um router comercial e implementar NAT
  + Experiência 5 – DNS
  + Experiência 6 – conexões TCP

**Aplicação de Download**

A aplicação deve receber como argumento da consola um link do tipo: **ftp://[<user>:<password>@]<host>/<url-path>** de acordo com a sintaxe descrita no RFC1738. Com este link devemos tentar conectar ao host indicado com as credenciais fornecias para fazer download do ficheiro indicado no url-path.

**Arquitetura**

O primeiro passo é então fazer parse do link recebido como argumento e para tal usamos a função **getArgsFromUrl** que recebe como argumentos o link e a struct a preencher com os valores lidos em caso de não haver erros.

De seguida recorremos à função **getIP** que, com o nome do host, obtém o IP do mesmo para passar à função **openSocket** que, por sua vez, cria e conecta um socket Berkely na porta 21.

Os próximos passos envolvem enviar e receber comandos do servidor e para tal usamos as funções **writeToSocket** e **readCommandFromSocket**. A **writeToSocket** recebe o file descriptor da socket que está conectada, o comando e a string extra, se existir. Os últimos 2 argumentos são escritos na socket, separados por um espaço. A função **readCommandFromSocket** recebe também o file descriptor da socket a ler e 2 variáveis para guardar o comando de resposta e um conteúdo da mesma. Primeiramente, utilizamos **usleep** para dar tempo ao servidor de responder (caso a conexão não seja das melhores). De seguida, escreve os 3 primeiros carateres na variável que guarda o comando (para se verificar se a resposta foi positiva ou negativa) e os restantes que existirem, na outra variável.

Usando então estas duas funções, depois de aberto o socket, é necessário fazer login usando os comandos “user example-user” e “pass exemple-pass”. Se o ficheiro não estiver no root do servidor é também necessário mudar o diretório atual do mesmo usando o comando “cwd diretório”.

De seguida é necessário entrar no modo passivo com o comando “pasv” e ler, da resposta, o IP do qual iremos ler o conteúdo do ficheiro. Para ler a resposta e obter o IP, a partir da mesma, usamos a função **getIPFromBody**, que guarda o IP e a porta das duas variáveis passados como argumentos. A seguir é criada outra socket que se conecta ao novo IP e porta.

Finalmente, enviando o comando “retr nome-do-ficheiro” para o primeiro socket aberto, podemos ler do novo socket o conteúdo do ficheiro para o criar.

**Resultados de um download com sucesso**

Se não houver erros no link passado à aplicação nem na conexão ao servidor pode-se observar no terminal outputs deste género:



Depois de alguns testes concluímos que a aplicação funciona como esperado, fazendo download de ficheiros de diferentes tipos e tamanhos, estando ou não no root do servidor.

**Configuração e análise da rede**

Partindo do princípio de que o switch tem as configurações default não há ligações entre nenhum dos aparelhos:

**Experiência 1 – Configurar uma rede IP**

O objetivo desta experiência foi ligar os computadores 3 e 4 ao switch, configurar à interface de rede eth0 de ambos com o IP e máscara corretos para que se conseguisse mandar mensagens entre os dois computadores.

Principais comandos:

* “ifconfig <interface de rede> <IP>/<máscara>” - atribuição de IPs a uma interface de rede
* “ping <IP destino>” - verificar a conectividade entre os dois computadores
* “route -n” e “arp -a” - verificar as tabelas de *forwarding* e ARP, respetivamente
* “arp -d <IP>” - apagar um IP da tabela de ARP

Questões sobre esta experiência:

* O que são os pacotes ARP e para que são usados?

ARP (Address Resolution Protocol) é um protocolo definido pela RFC 826 e tem como função mapear o endereço físico de uma máquina (MAC) na rede ao seu endereço lógico (IP). Para tal primeiro é feito um pedido em *broadcast* (à rede toda), enviando um pacote ARP a perguntar o endereço MAC da máquina com o endereço de IP pretendido. A máquina com o endereço lógico indicado irá enviar um pacote ARP de resposta em *unicast* (com apenas um destino) para a máquina que fez inicialmente o pedido com o seu endereço MAC. No fim, ambos guardam os dados um do outro em cache (tabela ARP).

* Quais são os endereços MAC e IP dos pacotes ARP e porquê?

Como quando o gnu3 tenta enviar um pacote ao endereço lógico do gnu4, não sabe qual o endereço físico deste último, envia então um pacote ARP em *broadcast* com o seu endereço de IP e MAC.

O pacote ARP de pedido contém o IP (172.16.10.1) e o endereço MAC (00:21:5a:5a:7d:16) como *source* para que quem responder ao pedido poder também guardar estes valores em cache. Como *destination* contém o IP (172.16.10.254) e MAC (ff:ff:ff:ff:ff:ff) sendo que se sabe obviamente qual o endereço lógico da máquina de destino (indicado no comando *ping*), mas não o físico.

O pacote ARP de resposta vem do gnu4 e contém como *destination*  os endereços que o pacote de pedido tinha como source e, por sua vez, como *source* tem o endereço de IP (172.16.10.254) e MAC (00:21:5a:5a:7d:3f) indicando então o seu endereço lógico ao gnu3.

Os 6 primeiros bytes do pacote são contém o endereço MAC da *destination* e os próximos 6 são o endereço MAC da *source*.

* Que pacotes gera o comando *ping*?

Se o endereço indicado no comando não estiver guardado na tabela ARP, gera primeiro um pacote ARP de *request*. Depois disso gera e envia pacotes ICMP (Internet Control Message Protocol).

* Quais são os endereços MAC e IP dos pacotes gerados pelo comando *ping*?

Os pacotes gerados no gnu3 e recebidos no gnu4 (pacotes *request*) têm como *destination* os endereços IP (172.16.10.254) e MAC (00:21:5a:5a:7d:3f) e como *source* os endereços IP (172.16.10.1) e MAC (00:21:5a:5a:7d:16).

Os pacotes gerados no gnu4 e recebidos no gnu3 (pacotes *reply*) têm como *destination* os endereços IP (172.16.10.1) e MAC (00:21:5a:5a:7d:16) e como *source* os endereços IP (172.16.10.254) e MAC (00:21:5a:5a:7d:3f).

* Como determinar se uma *frame* Ethernet é ARP, IP ou ICMP?

O tipo de identifica-se verificando, no *header* da mesma, os bytes 13 e 14 da *frame*. Se tiver o valor 0x0806 é um pacote ARP e tiver o valor 0x0800 é um pacote IP. Considerando que o protocolo ICMP é integrante do protocolo IP estes diferenciam-se pelo byte 24, se tiver o valor 0x01 é ICMP. Os bytes foram contados começando por 1 e não por 0 ao analisar os pacotes capturados pelo software wireshark.

* Como determinar o tamanho de uma *frame* recebida?

O comprimento total da trama pode ser visualizado recorrendo ao software wireshark. Pode-se ver diretamente o tamanho duma *frame* mas, analisando melhor reparamos que os 12 primeiros bytes correspondem aos endereços MAC do destino e fonte do pacote, por ordem, os próximos 2 indicam o tipo de pacote, e o restante é conteúdo do protocolo.

* O que é a interface *loopback* e porque é importante?

A interface *loopback* é uma interface de rede virtual que permite a um dispositivo de comunicação enviar pacotes destinados a si mesmo para identificar problemas na rede. Caso haja algum problema de comunicação, utiliza-se *loopback* para entender se o problema está no outro dispositivo (se receber a *frame* enviada a si mesmo) ou na rede (se não receber a *frame* enviada a si mesmo).

**Experiência 2 – Implementar duas virtual LANs num switch**

O objetivo desta experiência foi criar duas LANs virtuais, vlan10 e vlan11, adicionar os gnu3 e gnu4 à vlan10, o gnu2 à vlan11 e testar a comunicação entre eles todos.

Comandos principais:

* “ifconfig” – usado novamente para configurar a interface eth0 do gnu2
* Todos os comandos indicados na resposta à primeira pergunta desta experiência que indicam com criar uma vlan.

Questões sobre esta experiência:

* Como configurar uma vlan (no nosso caso vlan10 e vlan11)?

Para criar uma vlan é apenas necessário introduzir os seguintes comandos no GTKTerm do computador ligado à consola do switch:

* + configure terminal
  + vlan 10
  + end

Repetir os passos para a vlan 11.

Para adicionar um computador a determinada vlan é primeiro necessário que este tenha a porta eth0 ligada ao switch e que esteja configurada. Depois introduz-se os seguintes comandos no GTKTerm:

* + configure terminal
  + interface fastethernet 0/y
  + switchport mode access
  + switchport access vlan xx
  + end

Substituir “y” pelo número da porta no switch em que está ligada a porta eth0 do computador em questão e substituir “xx” pela vlan à qual se pretende adicionar o computador.

* Quantos domínios de *broadcast* existem e como deduzir isso dos *logs* obtidos?

Existem 2 domínios de *broadcast*, um com o gnu3 e o gnu4 e outro apenas com o gnu2. Podemos concluir isso a partir dos *logs* porque o *ping* enviado em *broadcast* a partir do gnu3 chega ao gnu4 mas não ao gnu2.

**Experiência 3 – Configurar um router em Linux**

**Experiência 4 – configurar um router comercial e implementar NAT**

**Experiência 5 – DNS**

**Experiência 6 – Conexões TCP**